HarvardX Data Science: Capstone

Rating Prediction on MovieLens

Padmashri Saravanan

May 7th 2020

Contents

1 Introduction

Having high quality consumer-grenerated ratings has become more important than ever, and an important and anticipated aspect of purchasing. Thus, it has also become essential for brand and retailer marketing strategies, such as Amazon, Google Play Store applications or food delivery company, Zomato. Companies then use these ratings to collect data and predict future ratings from a consumer, known as recommendation systems. This is consequently used to recommend products with high ratings to consumers.

In this case, we collect ratings from movies to form models and create a recommendation system using the 'MovieLens' dataset, collected by GroupLens Research.

1.1 Objective

The goal in this project is to predicts User Ratings (from 0.5 to 5) by training a machine learning algorithm that uses the inputs of a provided subset (the edx dataset set by the Capstone course) to predict Movie Ratings in a already given validation set.

The Root Mean Square (RMSE) is used to evaluate the performance of the algorithm and is frequently used to measure accuracy and the differences between the model and the actual values. A lower RMSE proves greater accuracy. RMSE is considered to be sensitive to outliers since each error on RMSE is proporitional to the size of the squared error, making the outliers have an abnormally large effect on the RMSE value.

We use the following formula to calculate the RMSE value:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,m}(\hat{y}_{u,m} - y_{u,m})^2}$$

In this project, we develop 4 separate models to compare the RMSE result and the one with the best result will be used to predict the ratings.

1.2 Data

The MovieLens dataset is given in the form of 'edx' and 'validation' subsets from the staff through RDS files.

```r
#############################################################
# Importing edx set and the validation set
#############################################################
# Note: this process could take a couple of minutes for loading required package:
# tidyverse and package caret
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")

edx <- readRDS("~/Downloads/edx.rds")
validation <- readRDS("~/Downloads/validation.rds")
```

The algorithm will be carried out only on the 'edx' subset and we will test the final algorithm on the 'validation' dataset.

2 Analysis

2.1 Data Analysis

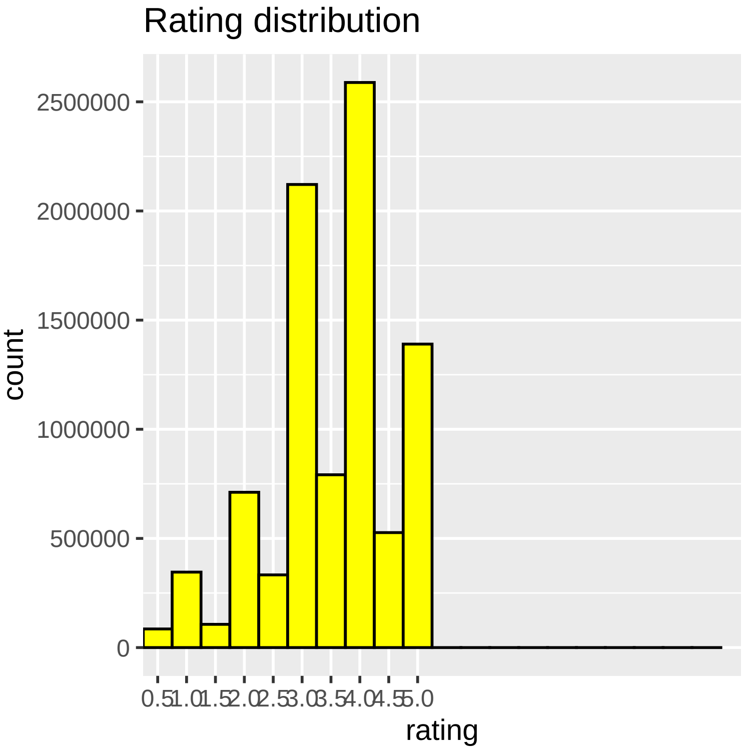
We first take a look at the 'edx' dataset.

```r
head(edx) %>%
  print.data.frame()
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

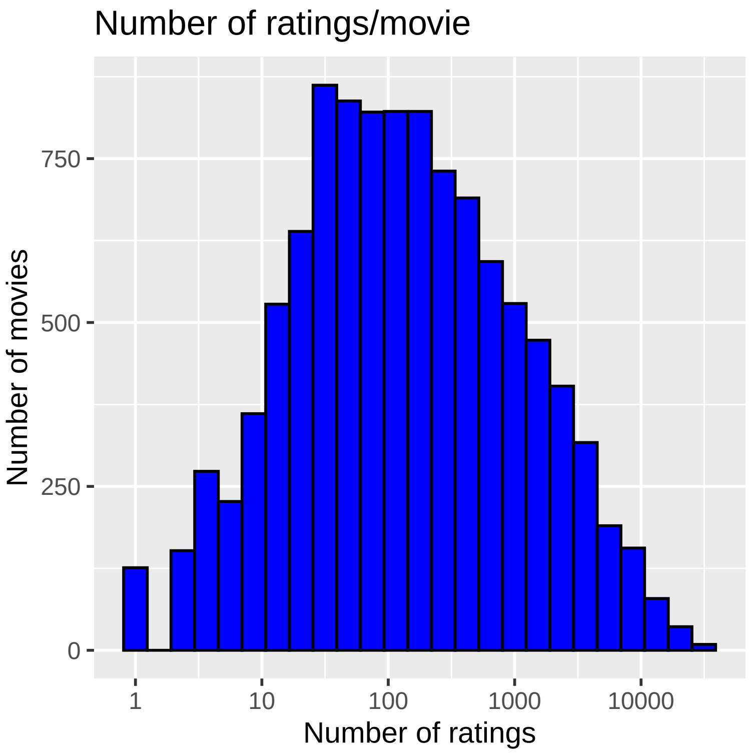
It contains 6 variables 'userID', 'movieID', 'rating', 'timestamp', 'title' and 'genres' and essentially shows the rating from a consumer for a single movie.

We also check if users gave a higher rating than a lower one by plotting a distribution.

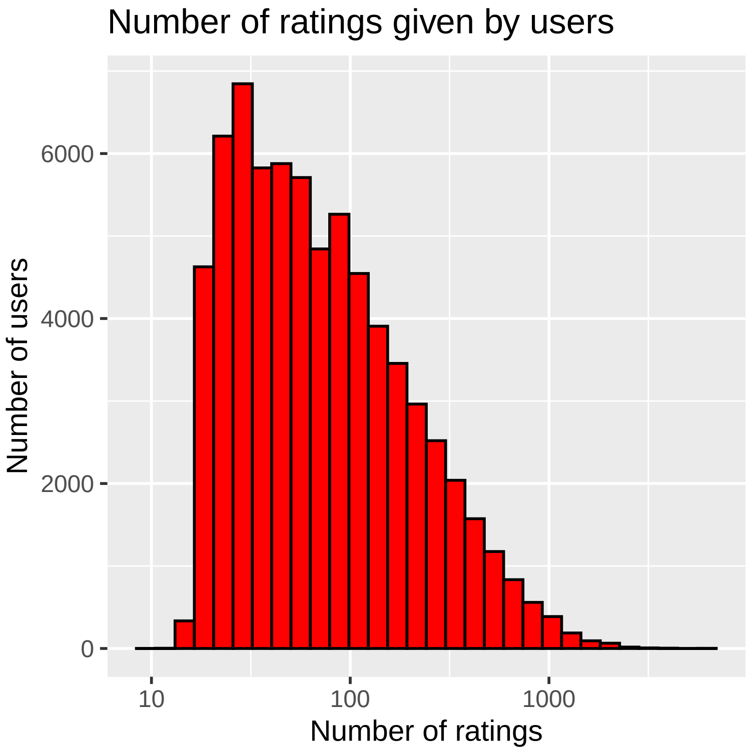


It is clear that 4 is the most common, afterwhich 3 and 5 are the most common and 0.5 is the least in count. We can also notice that whole ratings are more common than half ratings.

In this project, we also use regularisation and a penatly term to take into consideration that some movies have been rated alot more that another and some have only a single rating, giving an unreliable estimate of RMSE results for predictions.



Number of ratings/movie



Number of ratings given by users

Regularization is used to decrease the errors by avoid overfitting and fitting a function on the given training set and this also tunes the function through the addition of a penalty term, which compensates for the fluctuating function and avoids values that are extreme.

2.2 Modelling the Data

RMSE, as mentioned earlier, is calculated using the following definition where N is the number of combinations of users and movies.

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,m}(\hat{y}_{u,m} - y_{u,m})^2}$$

If the RMSE we obtain is greater than 1, it implies that the average error is greater than a one star rating, which makes our results unreliable.

We use the following function to calculate the values of RMSE for the given ratings and their respective predictions:

```
RMSE <- function(rating_actual, rating_prediction){
  sqrt(mean((rating_actual - rating_prediction)^2))
}
```

2.2.1 Modelling the mean movie rating

This model computes the dataset's average rating, which is between 3 and 4 as seen above. We predict by assuming the same rating for all movies and considering random variation for the differences.

$$Y_{u,m} = \mu + \epsilon_{u,m}$$

with $\epsilon_{u,m}$ as the independent error and μ being the *actual* movie rating. The estimate minimizing the RMSE is the least square estimate (LSE) of $Y_{u,m}$. in this case, is the mean of all ratings.

```
avg <- mean(edx$rating)
avg
```

```
## [1] 3.512465
```

The first RMSE can be calculated if we predict all unknown ratings with μ or avg:

```
rmse1 <- RMSE(validation$rating, avg)
rmse1
```

```
## [1] 1.061202
```

We then display the results in a table:

method	RMSE
Mean Movie Rating	1.061202

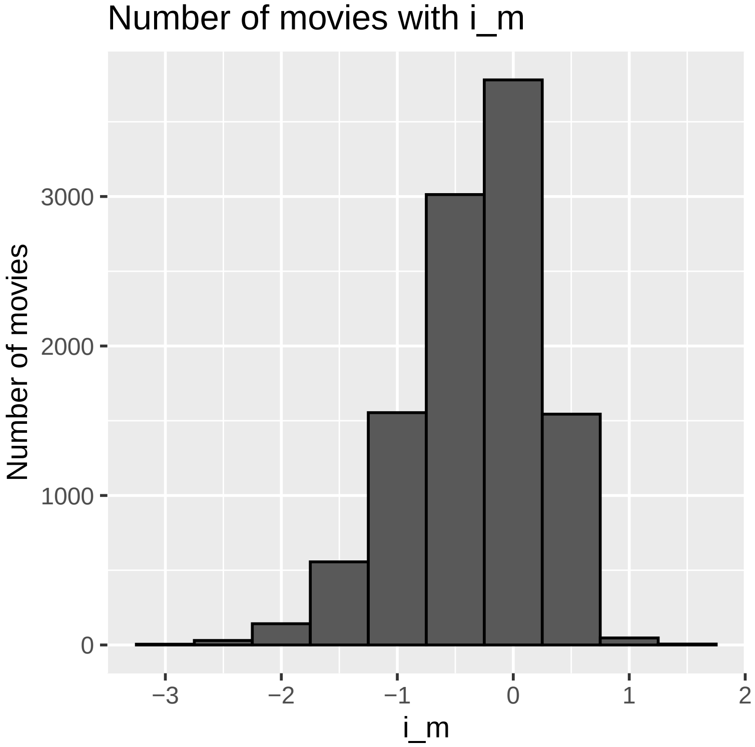
This gives us the foundation RMSE value to compare with the upcoming models.

2.2.2 Modelling the impact of movies

To refine the initial model, we consider how there are some movies with a higher rating than the rest because of its popularity. We do this by calculating the approximated deviation of each mean rating from the total mean μ. We define the resulting value as "i" for impact for each movie "m", i_m, which tells us the average ranking for movie m:

$$Y_{u,m} = \mu + i_m + \epsilon_{u,m}$$

Plotting this on a histogram we can observe that it is skewed to the left, that is, more movies have a negative impact.



We define this is as the movie impact penalty.

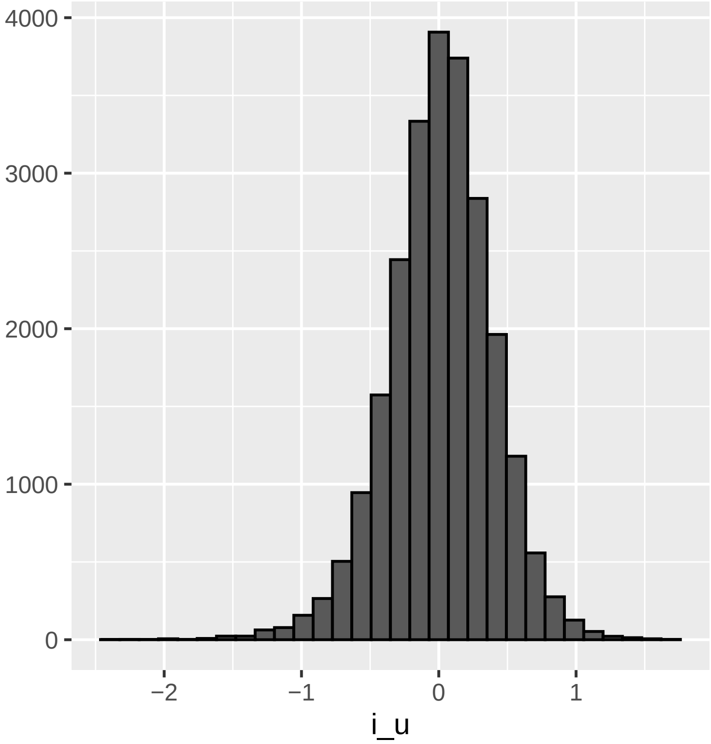
Our prediction also improves once we use this model.

method	RMSE
Mean Movie Rating	1.0612018
Modelling Movie Impact	0.9439087

We can see that there is an improvement in the RMSE value, but this model does not count the impact of the individual users.

2.2.3 Modelling the impact of movies and users

We compute the average rating for users for that have rated more than 100 movies, and define it as the user impact penalty.



This then results in the following improved model:

$$Y_{u,m} = \mu + i_m + i_u + \epsilon_{u,m}$$

where i_u is the impact created by the user. If a user that overcriticises ($i_u < 0$) rates a good movie ($i_m > 0$), the impacts contradict and we use this to predict that the user gave the good movie a lower rating than what it actually should have received.

We estimate the calculation by finding μ and i_m, and approixmating i_u, as the mean of

$$Y_{u,m} - \mu - i_m$$

We then further compute predictors and see that the RMSE value improves:

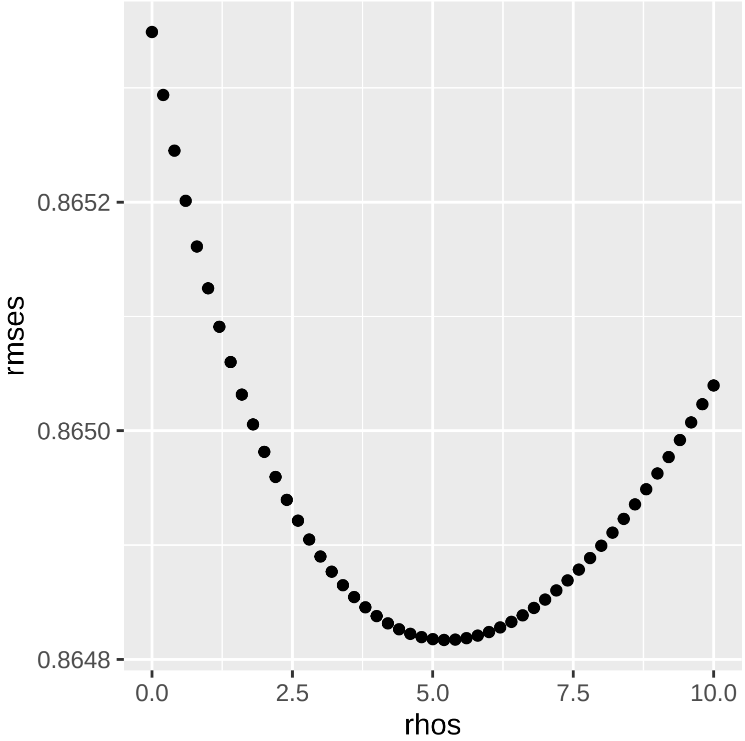
method	RMSE
Mean Movie Rating	1.0612018
Modelling Movie Impact	0.9439087
Modelling Movie and User Impact	0.8653488

2.2.4 Modelling the regularized movie and user impacts

Regularization helps us eliminate the possibility that the estimates of i_m and i_u are a result of movies with limited ratings and limited number of users rating it. We now define a tuning paramter ρ that will minimize the RMSE value and find the optimal value.

```r
rhos <- seq(0, 10, 0.2)
rmses <- sapply(rhos, function(k){

  avg <- mean(edx$rating)
  i_m <- edx %>%
    group_by(movieId) %>%
    summarize(i_m = sum(rating - avg)/(n()+k))

  i_u <- edx %>%
    left_join(i_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(i_u = sum(rating - i_m - avg)/(n()+k))

  rating_prediction <-
    validation %>%
    left_join(i_m, by = "movieId") %>%
    left_join(i_u, by = "userId") %>%
    mutate(pred = avg + i_m + i_u) %>%
    pull(pred)

  return(RMSE(rating_prediction, validation$rating))
})
```

We plot RMSE vs ρ to select the optimal ρ:



The optimal ρ for the complete model then becomes:

```
  rho_min <- rhos[which.min(rmses)]
rho_min
```

```
## [1] 5.2
```

The optimal rho for the full model is: 5.2

The new results will then be:

method	RMSE
Mean Movie Rating	1.0612018
Modelling Movie Impact	0.9439087
Modelling Movie and User Impact	0.8653488
Modelling Regularized Movie and User Impacts	0.8648170

3 Results

The RMSE values of all the models computed are presented below:

method	RMSE
Mean Movie Rating	1.0612018
Modelling Movie Impact	0.9439087
Modelling Movie and User Impact	0.8653488
Modelling Regularized Movie and User Impacts	0.8648170

Therefore, we found the lowest value of RMSE that is 0.8648170.

4 Discussion

Thus, the following is the final model of the project and proves to be efficient:

$$Y_{u,m} = \mu + i_m + i_u + \epsilon_{u,m}$$

5 Conclusion

We affirm to have built a machine learning algorithm to predict movie ratings with the *MovieLens* dataset. The regularized model including the impact of users is distinguished by the lowest RMSE value (0.8648170) and is hence the optimal model to use for the present project. We could also refine the RMSE value further by considering other impacts (genre, etc).

6 Appendix - Enviroment

```
## [1] "Operating System:"
```

```
##                _
## platform       x86_64-apple-darwin17.0
## arch           x86_64
## os             darwin17.0
## system         x86_64, darwin17.0
## status
## major          4
## minor          0.0
## year           2020
## month          04
## day            24
## svn rev        78286
## language       R
## version.string R version 4.0.0 (2020-04-24)
## nickname       Arbor Day
```